Assignment 2

(Last Update: 7 Oct)

Introduction

Download the code for this assignment here (updated 7 Oct 11:30am) and then unzip the archive.

As in assignment 1, this assignment uses python 3. Do not use python 2. You can work on the assignment using your favorite python editor. We recommend VSCode.

Post any questions or issues with this assignment to our discussion forum. Alternatively you may also contact your TA directly.

This assignment uses autograding for Problems 1 and 3. For the other problems please carefully check your implementation before submission.

Problem 1: Random Pacman play against a single random Ghost

In this part of the assignment you are going to implement

- a parser to read a pacman layout file in the file `parse.py`, and
- a game of pacman where pacman and ghost select moves randomly in the file `a1.py`.

Both these python files have already been created for. Do not change anything that has already been implemented. Our autograder relies on the existing code.

Start by implementing the `read_layout_problem()` in the file `parse.py`.

```
def read_layout_problem(file_path):
    #Your p1 code here
    problem = ''
    return problem
```

You can test your code with the first layout as follows.

```
python parse.py 1 1
```

This will supply the `test_cases/p1/1.prob` file as an argument to the function. The file has the following pacman layout.

```
seed: 8
%%%%
% W%
%  %
%  %
% .%
%P.%
```

```
%%%%
```

The first line is a random seed which will be used to initialize python's random number generator via the `random.seed()` function. This ensures that python generates a fixed sequence of random values. More on this later. The rest of the file is the pacman layout that you will have to parse. You can expect the following characters in the file.

```
'%': Wall
'W': Ghost
'P': Pacman
'.': Food
' ': empty Square
```

You may assume the layout is always rectangular surrounded by walls and that there is at least one ghost. For problem 1 there will be a single ghost only. Later Pacman we will have to deal with more ghosts (`'X'`, `'Y'`, `'Z'`). There will always be at least one food and there will always be exactly one pacman.

As in assignment 1, you can choose any data structure and return it from your `read_layout_problem` function. Once you are done with the parsing you can move on to the second part of this problem and implement the `random_play_single_ghost()` function in the file `p1.py`.

A correct implementation will return the following string for the first test case.

```
seed: 8
0
%%%%
% W%
%  %
%  %
% .%
%P.%
%%%%
1: P moving E
%%%%
% W%
%  %
%  %
% .%
% P%
%%%%
score: 9
2: W moving W
%%%%
%W %
%  %
%  %
```

```
% .%
% P%
%%%%
score: 9
3: P moving W
%%%%
%W %
%  %
%  %
% .%
%P %
%%%%
score: 8
4: W moving E
%%%%
% W%
%  %
%  %
% .%
%P %
%%%%
score: 8
5: P moving E
%%%%
% W%
%  %
%  %
% .%
% P%
%%%%
score: 7
6: W moving S
%%%%
%  %
% W%
%  %
% .%
% P%
%%%%
score: 7
7: P moving N
%%%%
%  %
% W%
%  %
% P%
%  %
%%%%
```

```
score: 516
WIN: Pacman
```

As you can see, Pacman and the Ghost make moves alternatively. Pacman starts by making a move east. This is determined using the random.choice() function on the available moves to pacman. In this case for the start state pacman can move East (E) for the food or North(N) for the empty square. Pacman (P) moves east. Let's reproduce that decision to understand the sequence of moves generated here.

```
(base) scdirk@Dirks-Air a2 % python
>>> import random
>>> random.seed(8, version=1)
>>> random.choice(('E', 'N'))
'E'
```

Next, the Ghost (`W`) moves West (`W`). The available actions to the ghost are `W` and `S`.

```
(base) scdirk@Dirks-Air a2 % python
>>> import random
>>> random.seed(8, version=1)
>>> random.choice(('E', 'N'))
'E'
>>> random.choice(('S', 'W'))
'W'
```

Important: You must ensure that the parameter to the random.choice() function is sorted alphabetically. Otherwise you will not be able to reproduce the exact result and you won't be able to pass the autograder.

In the test case 1 above, Pacman won because he cleared all the food. Pacman and Ghost may move to any square except a wall. A ghost may move on top of a food but it won't eat the food. If Pacman is eaten, the game is over and Pacman loses. Here is another run where that happens. This is test case 2.

```
seed: 42
0
%%%%
%.W%
%  %
%  %
% .%
%P.%
%%%%
1: P moving E
%%%%
%.W%
```

```
%  %
%  %
% .%
% P%
%%%%
score: 9
2: W moving S
%%%%
%. %
% W%
%  %
% .%
% P%
%%%%
score: 9
3: P moving W
%%%%
%. %
% W%
%  %
% .%
%P %
%%%%
score: 8
4: W moving N
%%%%
%.W%
%  %
%  %
% .%
%P %
%%%%
score: 8
5: P moving E
%%%%
%.W%
%  %
%  %
% .%
% P%
%%%%
score: 7
6: W moving S
%%%%
%. %
% W%
%  %
% .%
```

```
% P%
%%%%
score: 7
7: P moving N
%%%%
%. %
% W%
%  %
% P%
%  %
%%%%
score: 16
8: W moving W
%%%%
%. %
%W %
%  %
% P%
%  %
%%%%
score: 16
9: P moving W
%%%%
%. %
%W %
%  %
%P %
%  %
%%%%
score: 15
10: W moving S
%%%%
%. %
%  %
%W %
%P %
%  %
%%%%
score: 15
11: P moving E
%%%%
%. %
%  %
%W %
% P%
%  %
%%%%
score: 14
```

```
12: W moving S
%%%%
%. %
%  %
%  %
%WP%
%  %
%%%%
score: 14
13: P moving S
%%%%
%. %
%  %
%  %
%W %
% P%
%%%%
score: 13
14: W moving E
%%%%
%. %
%  %
%  %
% W%
% P%
%%%%
score: 13
15: P moving N
%%%%
%. %
%  %
%  %
% W%
%  %
%%%%
score: -488
WIN: Ghost
```

Scoring is done as follows.

```
EAT_FOOD_SCORE = 10
PACMAN_EATEN_SCORE = -500
PACMAN_WIN_SCORE = 500
PACMAN_MOVING_SCORE = -1
```

You may define any number of your own functions to solve the problem.
Once you are done you can check if you pass all the test cases for Problem 1.

```
(base) scdirk@Dirks-Air a2 % python p1.py -6
Grading Problem 1 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
```

You may import anything from the Python Standard Library. Do not import packages that are not included in Python such as numpy.
Make sure that you pass all provided test cases before moving on to the next question. Note that we may use novel test cases for marking. You can also design your own new test cases for testing.

Problem 2: Pacman play against a single random Ghost 

Next you will write a simple reflex agent that does not move randomly. Instead it will move to a better position based on a simple evaluation function, designed by you, that takes a state-action pair and returns the best action. Hint: You may consider evaluating the distance to the closest ghost and combine somehow with the distance to the closest food.

You may reuse some of the code that you wrote in Problem 1 and implement just the evaluation function and action selection by completing the code in the function `better_play_single_ghosts()`. This function should return two values, the gameplay as usual and the winner which should be `'Pacman'` if Pacman wins. You can see how the `better_play_single_ghosts()` is used in the main function of the script `p2.py`.

This time, we will have no autograder and ghosts should move randomly without a seed. You may inspect the test cases and note that the seed is -1, which indicates that there will be no seed. This allows us to conduct multiple trials.

```
(base) scdirk@Dirks-Air a2 % python p2.py 1 10 0
test_case_id: 1
num_trials: 10
verbose: False
time:  0.09921669960021973
win % 90.0
```

The three parameters control the test case id, number of trials and a verbose option that will output the actual gameplay if it is set to `1` instead of `0`.

```
base) scdirk@Dirks-Air a2 % python p2.py 1 1 1
test_case_id: 1
num_trials: 1
verbose: True
seed: -1
```

```
0
%%%%%
% . %
%.W.%
% . %
%. .%
%   %
%  .%
%   %
%P .%
%%%%%
...
123: P moving W
%%%%%
%   %
%   %
%   %
%P  %
%   %
%   %
%   %
%W  %
%%%%%
score: 518
WIN: Pacman
time:  0.008844137191772461
win % 100.0
```

With a reasonable evaluation function you should be able to win half of the games easily. No need to spend too much time on this question. You will implement expectimax soon and play better games.

Problem 3: Random Pacman play against up to 4 random Ghost

This problem is similar to problem 1 except that you will implement a game against multiple ghosts.

Ghosts cannot move on top of each other. If a Ghost is stuck no move will be done. Pacman will start followed by `W`, `X`, `Y`, `Z`. Note that the last three ghosts are optional. So the following combinations are possible:

- 1 Ghost: `W`
- 2 Ghosts: `W, X`
- 3 Ghosts: `W, X, Y`
- 4 Ghosts: `W, X, Y, Z`

Here is a game with 2 ghosts.

```
seed: 42
```

```
0
%%%%
%.X%
%W %
%P %
%%%%
1: P moving E
%%%%
%.X%
%W %
% P%
%%%%
score: -1
2: W moving E
%%%%
%.X%
% W%
% P%
%%%%
score: -1
3: X moving W
%%%%
%X %
% W%
% P%
%%%%
score: -1
4: P moving N
%%%%
%X %
% W%
%  %
%%%%
score: -502
WIN: Ghost
```

Note that ghosts move in alphabetical order, i.e., `W` first followed by `X` etc.

Problem 4: Pacman play against up to 4 random Ghost

This problem is similar to P2 except that we have multiple ghosts as in P3. Again, no autograding is provided and you may check the performance of your agent as follows.

```
(base) scdirk@Dirks-Air a2 % python p4.py 1 100 0
test_case_id: 1
num_trials: 100
verbose: False
time:  0.3293917179107666
```

```
win % 80.0
```

Don’t worry too much if the performance of your agent is much worse compared to P2. This is to be expected considering the problem difficulty has increased with multiple ghosts.

Problem 5: Minimax Pacman play against up to 4 minimax Ghosts

In this problem, you will implement a minimax agent and minimax ghosts. Your minimax should search until a ply depth k (i.e., k moves by everyone) and then use an evaluation function to determine the value of the state. The depth is provided as a parameter to the function. You can test the performance of your game playing agent as follows.

```
(base) scdirk@Dirks-Air a2 % python p5.py 1 4 10 0
test_case_id: 1
k: 4
num_trials: 10
verbose: False
time:  0.93696603775024414
win % 60.0
```

Problem 6: Expectimax Pacman play against up to 4 random Ghosts

Finally, you will implement an expectimax agent against random ghosts. Your expectimax should search until a depth k and then use an evaluation function to determine the value of the state. The depth is provided as a parameter to the function. You can test the performance of your game playing agent as follows.

```
(base) scdirk@Dirks-Air a2 % python p6.py 1 4 10 0
test_case_id: 1
k: 4
num_trials: 10
verbose: False
time:  0.2952871322631836
win % 90.0
```

To submit your assignment to Moodle, `*.zip` the following files ONLY:

- `p1.py`
- `p2.py`
- `p3.py`
- `p4.py`
- `p5.py`
- `p6.py`
- `parse.py`

Do not zip any other files. Use the `*.zip` file format. Make sure that you have submitted the correct files.